March 2019

Detection and Estimation of a Plume with an Unmanned Terrain Vehicle

Daniel C. Weber
Worcester Polytechnic Institute

Katherine Elizabeth Burris
Worcester Polytechnic Institute

Spencer Wynn Herrington
Worcester Polytechnic Institute

Theresa Bender
Worcester Polytechnic Institute

Detection and Estimation of a Plume with an Unmanned Terrain Vehicle

A Major Qualifying Project Report
Submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in Partial Fulfillment of the Requirements for the
Degree of Bachelor of Science
in Aerospace Engineering

by

Theresa Bender

Theresa Bender

Katherine Burris

Spencer Herrington

Daniel Weber

March 1, 2019
Approved by:

Michael A. Demetriou, Advisor
Professor, Aerospace Engineering Program
WPI

Abstract

This project includes the research, design, construction, and implementation of a gas-sensing autonomous terrain vehicle robot. The goal of this project is to localize a plume of CO_2 gas using CO_2 sensors onboard the robot. The robot is controlled by a Raspberry Pi, an onboard inertial measurement unit (IMU), and magnetic rotary encoders. To implement a gradient ascent guidance policy, the robot receives measurements from the CO_2 sensors and computes the angle of highest concentration to turn toward the source. An Extended Kalman Filter is used to estimate the position of the robot while in motion, using data from the encoders and IMU. A mass flow controller and sparger diffuser are used to dependably simulate the point source plume. This work on plume detection and estimation has applicability for all unmanned vehicles tracking a multitude of plumes in indoor and outdoor environments.

Acknowledgements

This project would not have been possible without the help and support of everyone who collaborated us throughout the entire project process. More specifically, we would like to thank our advisor Professor Demetriou for his help and guidance throughout the entirety of this project. Additionally, we would like to thank Professor Blandino for assisting us in designing and building the point source for our plume generation system.

Table of Authorship

Section	Author
Introduction	SH
1.1	SH
1.2, 1.2.1, 1.2.2	TB, SH
1.3, 1.3.1, 1.3.2, 1.3.3, 1.3.4, 1.3.5, 1.3.6	SH
1.4, 1.5	TB
1.6	TB
1.7	TB, KB
1.8	SH
1.9	KB
Robot	DW
2.1, 2.2, 2.3	DW
2.4	SH
Plume Generation System	KB
3.1, 3.2, 3.3	KB
Control	TB
4.1	TB
4.2	TB, DW
4.3, 4.3.1, 4.3.2, 4.3.3, 4.3.4	TB, SH
4.4	DW
4.5	KB, DW
Summary	SH
Conclusions	TB, KB
Recommendations	TB
5.3.1, 5.3.2, 5.3.3	TB
5.3.4	DW
Appendix A	DW
Appendix B	TB, KB
Appendix C	DW
Appendix D	DW

Table of Contents

Table of Figures

1 Introduction

Gases are present in many locations on Earth, as well as on other planets that have been or are currently being explored. Mapping gas plumes has many applications, including monitoring Earth's atmospheric conditions, tracking significant geological events, and sensing the presence of certain gases. Humans have historically been the responders to the presence and threats of gases, however this is not always the most ideal or feasible option. While some gases can be harmless, others are harmful to humans, which complicates the detection and analysis of their activity. For example, CO_2 can cause suffocation, incapacitation, and unconsciousness [1]. The new age of autonomy provides another options for exploring and analyzing dangerous sites. For example, robots may be able to locate the source and severity of a carbon monoxide leak in a populated building. This ability to detect and map plumes of gas without any human interaction can help save human lives.

Another important application of autonomous detection and estimation of gas plumes is to advance human knowledge of other planets. Robotic exploration of the solar system provides a cost-effective and low-risk manner to acquire information, since human life is not at stake. For example, the Martian environment is harmful to humans, so utilizing robotic explorers such as Mars rovers enable planetary exploration without exposing humans to the dangers of space travel [2]. Robots sent to other planets can use sensors to determine the concentration of the atmospheric gases. They can also locate a possible point source of emitted gas and map the concentration in a certain area. This information provides valuable planetary knowledge about the state and evolution of the solar system. In addition, if humans decide to travel to another planet, gas-detection robots would be extremely useful in facilitating scientific exploration and securing astronaut safety.

In previous years, students have created a system to generate a plume of gas in a controlled environment, and then detect and map the gas plume with an autonomous robot. This autonomous robot is equipped with a microcontroller, inertial mass unit (IMU), motor encoders, carbon dioxide (CO_2) sensors, and a wireless communication module that relays data to a base station. The base station performs the calculations and communicates orientation and position data to the robot. This Major Qualifying Project (MQP) aims to advance the precision and results of the previous MQPs.

1.1 Background and Literature Review

The background and literature review section will review previous work related to this project. First, related work outside of WPI will be discussed, followed by a subsection discussing and analyzing work performed in this area at WPI, primarily through previous MQPs. The information collected in this section helped to provide the team with a starting point, as well as to guide the direction of the project.

1.2 Overview of Relevant Work Done in Field

This section will provide information of similar work performed outside of WPI, including Kalman Filter design and length scales. This information is necessary for designing and programming the robot for example, length scales need to be study to get a solid understanding about the gas diffusion. Gas diffusion needs to be known to make an accurate estimate of how far apart the CO_2 Sensors need to be in order to get accurate measurements.

1.2.1 Kalman Filter

A Kalman Filter is an algorithm used to estimate and predict the position of an autonomous vehicle based on sensor measurements [3]. For this project, the Kalman Filter is very useful in order to make sure the guidance system of the robot is correctly moving to the next position error

free. If the guidance system does have errors the team will use the Kalman Filter to correct these errors however small or large they may be.

1.2.1.1 Kalman Filter History

Predicting the future state of a system is important for many applications, particularly in the aerospace and robotics industry. The Wiener filter, published in 1949, predicted positions and filtered noise for time invariant systems to predict systems states. In 1960, Rudolf E. Kalman's paper introduced the Kalman Filter, which provided a significant upgrade to the Wiener filter by predicting random signals and separating random signals from random noise for time-varying linear systems. Following Kalman's publication, the chief of the Dynamic Analysis Branch at NASA Ames Research Center (ARC) invited Kalman to present his work at ARC in belief that it could help solve the trajectory estimation problem of sending Apollo astronauts to the moon and back [4]. For these estimations, linearization was used to approximate the estimated trajectory, a mechanism which later became known as the extended Kalman Filter. The Extended Kalman Filter proved to be an integral part of the guidance and navigation system for the Apollo mission, as it accurately estimated the Saturn V trajectories and successfully brought astronauts to and from the lunar surface [4].

The Kalman Filter continues to have a profound impact, especially in the aerospace and robotics industry, where it is used for guidance, navigation and control, signal processing, and autonomous robotic control. It can estimate quantities from the indirect measurement of a sensor, as well as estimate a state by combining measurements of various sensors. This makes it ideal for guiding missiles towards a moving target, estimating spacecraft trajectories, and moving robots based on sensor measurements. Kalman Filters enable novel uses of autonomy by allowing in-time control operations based on current data readings. This is particularly useful for astronautic

applications since it can eliminate the need for constant communication with spacecraft from far destinations where communication lags are significant.

1.2.1.2 Kalman Filter Overview

The concept behind the Kalman Filter is to input initial assumptions and noisy measurements so that it can remove or filter out the noise and uncertainty to provide the desired states. Kalman Filters can be used to estimate a quantity from the indirect measurement of a sensor, as well as to estimate a state by combining measurements of various noisy sensors. The Kalman Filter works using a prediction and update phase. During the prediction state, the previous state's position, velocity, and acceleration are used to determine the current position. The prediction starts at the previous position. Next, the velocity at the current state multiplied by the time step is added, since that equals the distance moved if the velocity was constant during that time step. Finally, the acceleration at the current state multiplied by the time step squared is added, as that equals the distance travelled during the time step as a result of the acceleration term. This result is the best prediction for the position of the system based on information from the previous state.

In the update phase, this prediction is compared with the sensor measurements and their variances to determine the how much each will factor into the final estimation state. The current state, sensor measurements, and confidence in the readings are used to provide the final updated state.

1.2.1.3 Extended Kalman Filter Overview

The Extended Kalman Filter has the same purpose as the Kalman Filter, but it is applied to nonlinear systems. In order to linearize the equations, Jacobian matrices are used at each time step. The Jacobian is the matrix of all first-order partial derivatives of a vector function, so it provides the best linear approximation of the function.

1.2.2 Length Scales

An important characteristic of the robot is where to place the carbon dioxide sensors to acquire an accurate representation of the CO_2 distribution. Ideally the flow will be laminar, and the distribution of CO_2 will have a constant gradient; however, the presence of eddies, swirls of fluid that differ in direction from the rest of the flow, preclude this ideality [4]. These eddies disrupt gas uniformity and can lead to inaccurate representations of the gas distribution if the eddy size is greater than the measurement scale. The gradient between two CO_2 sensors, a and b, can be modeled with the slope equation,

$$\text{slope} = \frac{f(b)-f(a)}{b-a}. \tag{1}$$

Due to the presence of eddies, it is important to determine the minimum distance ($b - a$) that will provide a meaningful estimation of the gas gradient. A common standard is that the distance between CO_2 sensors should be twice the size of the largest eddies.

Length scales in fluid mechanics quantify these eddies by their diameter, and they are a function of molecular viscosity, dissipation rate, and kinetic energy. The three primary length scales are the Kolmogorov scale, Taylor scale, and Integral scale, which characterize the smallest, intermediate, and largest eddies, respectively. Large eddies are unstable and break up into smaller eddies, and these smaller eddies break up into smaller eddies until they are stable and can dissipate kinetic energy [4]. The integral length scale is of importance to this project because large eddies can disrupt measurements.

1.3 Overview of Earlier Relevant Major Qualifying Projects

The “Detection and Estimation of a Plume with an Unmanned Terrain Vehicle” Major Qualifying Project has been an MQP since 2015, as four different groups have taken on the challenge of creating a plume generation system and developing an autonomous robot to detect

CO_2. In 2015 and 2016, the robot and plume generation system were separate MQPs. In the past two years, these two tasks have been combined into a single project. To begin the project, the team researched into the past six MQPs to acquire a better understanding of the project, as well as learn about the successes and shortcomings of previous years.

1.3.1 Summary of NAG 1501

This MQP Team designed a plume generation system and a plume detection system, as well as analyzed what gases could possibly work for this system. Because CO_2 is safe, easy to access, easy to refill, and cost effective, the team decided to use CO_2. This team performed many simulations and predictions of plume profiles to create the most effective system. They used these simulations to create the system so it would be effective in a practical environment [6].

1.3.2 Summary of MAD 1501

This MQP team used a Khepera IV robot because of its small size and maneuverability. The Khepera is a two-wheel drive robot that has two caster wheels for stability. The Khepera also has an IMU built into it, so the team did not have to add an accelerometer or gyroscope to obtain acceleration and angular rate data. The group used COZIR Ambient 10K CO_2 Sensor to detect the gas plume which used very little power (3.5mW) and had a low response time (three seconds). The team then wrote a program and performed simulations in the motion control of the robot, finding parameters of speed using the geometric properties of the Khepera IV [7].

1.3.3 Summary of NAG 1602

This MQP team created new simulations to verify the results of the previous NAG 1501. This team also bought a new mass flow controller and a new wind speed sensor. They created a LabView program to control the plume generation system and read sensor data. The wind speed

sensor was found to be inefficient in correctly mapping the plume of gas because of its low quality [8].

1.3.4 Summary of MAD 1601

Similar to the previous MAD 1501 team, this MQP team decided to use the Khephera IV and the COZIR CO_2 Sensors. The team analyzed the motion of the Khephera IV robot and defined the kinematic equations of motion of the system. Once the team found the kinematic equations of motion, they obtained the discrete time equations of motion. Discretization is a process that estimates a continuous time system in relation to discrete time values, so a processor can then evaluate the state of the robot as an iterative process. The team then represented these equations in state-space. Since the system of equations is non-linear, the team had to use an Extended Kalman Filter to approximate the linearization of the system. This team also created the sensor tower proposed by the MAD 1501 team [9].

1.3.5 Summary of MAD 1702

This team switched from the Khepera IV to the iRobot Create 2. The justification behind this decision was due to the Create's wider and heavier base, which made it less likely to tip over with the sensor tower like the Khepera did in previous MQPs. The problem with the Create is that it did not have an onboard sensor like the Khepera did, so they had to attach an IMU and an on-board processing unit (RaspberryPi). The team also created an Extended Kalman Filter for the new iRobot Create 2 [10].

1.3.6 Summary of MAD 1802

This MQP team decided to research new robots and chose a different style from previous MQPs. The team decided to use the Dagu Wild Thumper 4WD because of its stability and its

accessories that could be easily added to the top of the robot. The team also created a new sensor platform to hold the CO_2 sensors and chose new CO_2 sensors to use. The SprintIR WR 20% CO_2 sensors were chosen because the time it took the previous sensors to return to an ambient CO_2 reading (settling time) was very long, where the SprintIR settling time was much shorter. This team also created a new point source for the plume generation system [11].

1.4 Project Goals

The goals of the Detection and Estimation of a Plume with an Unmanned Terrain Vehicle Project are to:

- Develop a robot to autonomously navigate itself towards a CO_2 plume source using measurements from CO_2 sensors.
- Develop a plume generation system capable of producing a plume of CO_2 with a known concentration.
- Track the position of the robot as it navigates towards the CO_2 source using an Extended Kalman Filter, an algorithm that takes in data with unknown noise and compares it with the previous state to predict the future state of the robot.
- Compare the position of the robot taken with the EKF to that taken with AprilTags, a tracking system that uses the overhead camera.

1.5 Project Design Requirements, Constraints and Other Considerations

The design requirements of the unmanned terrain vehicle include:

- The robot shall be capable of rotation, forward, and backward motion, in order to meet the goal of navigation towards a plume source.
- The robot shall be equipped with four equally spaced CO_2 sensors.

    - The CO_2 sensors shall be at least 50 centimeters above ground level, in order to receive CO_2 readings without ground effect.
    - The CO_2 sensors shall be at least far enough apart from each other to provide a meaningful estimation of the gas gradient.
    - CO_2 sensors shall be chosen that minimize rise and settling time.
- The robot shall be capable of real-time communication with the ground station.
- The robot power system shall be capable of performing at least one full experiment (~5 minutes).
- The design requirements of the plume generation system include:
- The plume system shall emit CO_2 as close to a point source as possible.
- The CO_2 level shall be controllable by the MQP team.
- The design constraints of the project include:
- The system should cost less than $1000, the MQP budget.
- The power system shall not use LIPO batteries.

1.6 Project Management

This project team consisted of four team members: Theresa Bender, Katherine Burris, Spencer Herrington, and Daniel Weber. The primary tasks of the project included designing and building a robot, developing a plume generation system, implementing a program for autonomous movement of the robot towards the plume source, and tracking the robot through Extended Kalman Filter equations derived by the team.

The team met several times a week to troubleshoot the code, continue development of the plume system, perform tests on the robot, and integrate individual work done over the previous few days. The team met weekly with the advisor, Professor Demetriou, to provide updates and

receive guidance and advice for the project. Each week, a different member of the team presented a PowerPoint presentation to the advisor with the weekly progress and updates. These presentations abided by proper citing and copyright laws, as taught in the Aerospace Colloquium. In addition, the people and resources that provided help were accredited in the presentations and papers.

1.7 MQP Objectives, Methods and Standards

1. Create a robotic base to move autonomously and house CO_2 sensors.
    a. Design a robotic base with substantial stability.
    b. Design a sensor platform to place CO_2 sensors at least 50 cm of the ground.
    c. Create a base that is easy to manufacture and build.
    d. Program a controller to move the base with pre-determined movements.
2. Design a plume source to evenly diffuse CO_2 for the robot to detect.
    a. Develop a design that would efficiently diffuse CO_2 into the test area.
    a. Design and build a new diffuser to better emulate a point source.
    b. Test the plume with the CO_2 sensors to ensure it is diffusing evenly.
3. Make the robot autonomously navigate towards the CO_2 plume source.
    a. Develop a method that uses data from the CO_2 sensors to autonomously turn and navigate towards the highest CO_2 concentration.
    b. Program this method in a Python script to command the robot to drive forward, turn left and turn right.
    c. Test the program and accuracy of this method by comparing sensor readings and calculated values of the location of where the robot should turn to where the robot actually turned, as commanded by the program.

1.8 MQP Tasks and Timetable

In order to organize and plan the Project timeline the team constructed a Gantt Chart that roughly outlines the entire project. This chart was obviously updated throughout the three terms because the timeline changed dramatically for some of the different tasks, but some of the tasks did stay the same. Shown in Figure 1, Figure 2 and Figure 3 are the tasks for A, B and C terms respectively.

PROJECT TITLE	Detection and Estimation of a Plume	DATE	8/23/18 - 3/1/19
PROJECT TEAM	Theresa Bender, Katherine Burris, Spencer Herrington, Daniel Weber		

TASK TITLE	START DATE	END DATE	DURATION	PCT OF TASK COMPLETE
Background Research				
Research Previous MQP's	8/23/18	9/4/18	11	100%
Research Current Robot and Others	8/23/18	9/4/18	11	100%
Research Sensors and Microcontrollers	8/28/18	9/11/18	13	100%
Research Kalman Filters and Interpolation	8/28/18	9/25/18	27	100%
Research Length Scales	8/28/2018	9/30/18	32	100%
Fourier Transforms	9/18/18	10/9/18	21	100%
			0	0%
Testing Sensors and Hardware				
SprintIR CO_2 Sensor Testing	9/11/18	9/25/18	14	100%
IMU Testing	9/11/18	9/25/18	14	100%
Motor Testing	9/25/18	10/11/18	16	100%
			0	0%
Design and Build and Test of the Robot				
Design of Robot	8/28/18	3/1/19	183	100%
Build of Robot	9/25/18	3/1/19	156	100%
Testing of Robot	10/23/18	3/1/19	128	100%
Coding Guidance System	10/23/18	3/1/19	128	100%
Plume Testing	10/23/18	3/1/19	128	75%
Incorporating Sensors to Guide Robot	10/23/18	3/1/19	128	100%
MQP Drafts				
Draft 1	8/28/18	10/11/18	43	100%
Draft 2	10/23/18	12/14/18	51	100%
Final Draft	1/9/19	3/1/19	52	100%

A Term

WEEK 1 | WEEK 2 | WEEK 3 | WEEK 4 | WEEK 5 | WEEK 6 | WEEK 7 | WEEK 8

M T W R F M T W R F M T W R F M T W R F M T W R F M T W R F M T W R F M T W R F M T W R F

Figure 1: A Term Timetable.

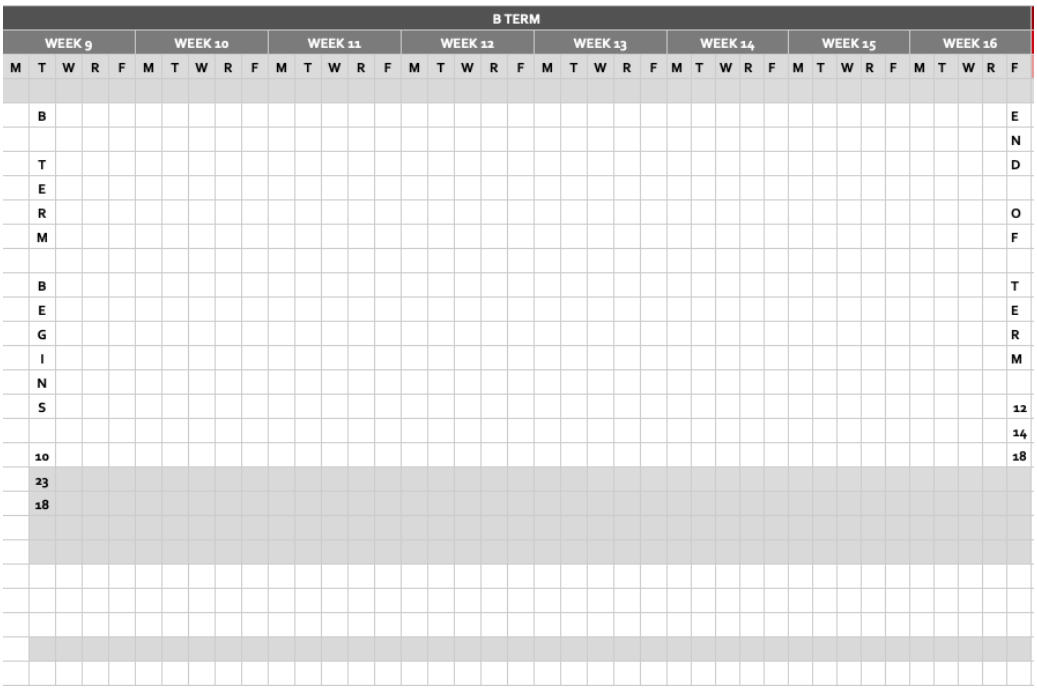

Figure 2: B Term Timetable.

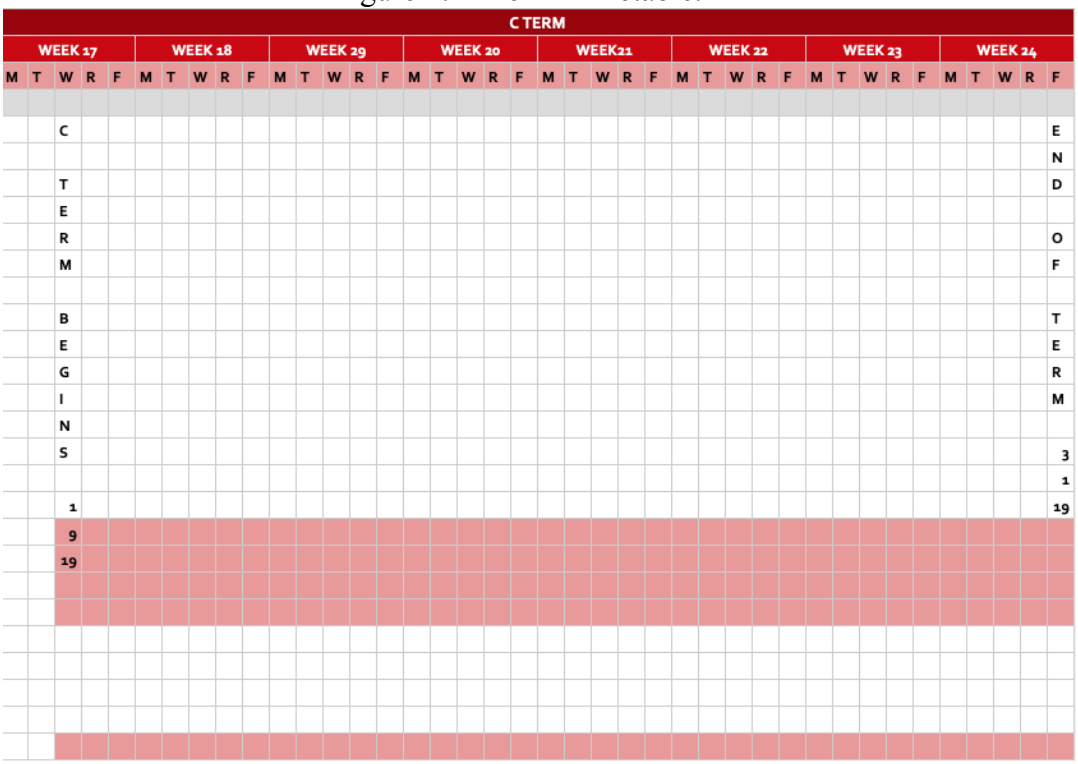

Figure 3: C Term Timetable.

1.9 Broader Impacts

Detection and estimation of a plume with an unmanned terrain vehicle allows for CO_2 or other harmful gases to be easily detected without the need for human contact with the gas. This robot would allow people to find the location of a gas leak and the concentration of that gas, all while observing from a safe distance. In the future, the robot could be fitted with sensors that are able to detect other harmful gases.

While these robots can be lifesaving, they are not environmentally friendly. More specifically, designing, building, testing, and ultimately trashing a robot to move onto the next model is a waste of resources such as energy, rare metals, and many times results in electronic waste [12]. This MQP project has been in effect for multiple years, and every year there has been testing with the plume source, allowing for CO_2 to be pumped into the atmosphere which is one of the greenhouse gases that is greatly contributing to climate change [13].

2 Robot

Upon reviewing the status of the robot from MAD 1802, the team decided to research new robotic platforms. This section details the design and criterion of a robotic base suitable for this application. In researching commercial off-the-shelf robots, there were not many complete robotic platforms that satisfied the project budget and requirements. Many would have experienced similar problems faced in previous years, such as poor stability and difficulty in adapting the base for this application.

2.1 Design Considerations of Robot

Reaching the conclusions of the research into robotic platforms as mentioned previously, the team decided to design the robot. The robot designed is based on that of the iRobot Create and Khepera IV robots to simplify the equations of motion in making fewer assumptions about the robot's motion, namely the scrub steering four-wheel drive.

SolidWorks® was used to design the new robotic base, in order to ensure proper sizing and space claim of components. This model was also used to create the 3D printed parts and laser cut patterns to build the robot. The complete CAD model with all components is shown in Figure 4. A list of components can be found in Appendix A.

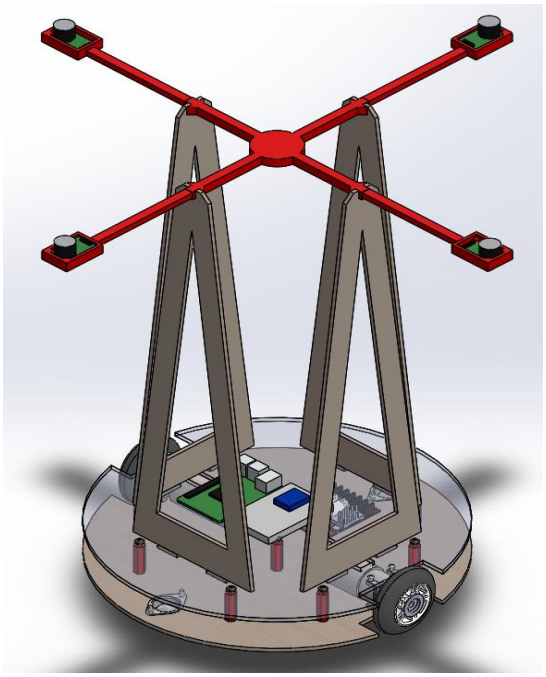

Figure 4: SolidWorks® Representation of Robot.

The robot has a 14-inch diameter base made from laser cut ¼ inch plywood. The purpose of the large base was to eliminate the possibility of tipping with a large sensor platform attached. The use of a laser cutter allowed for the creation of the exact dimensions outlined in the CAD, including custom mounting patterns for the various components. The pattern can be seen in Figure 5 and Figure 6.

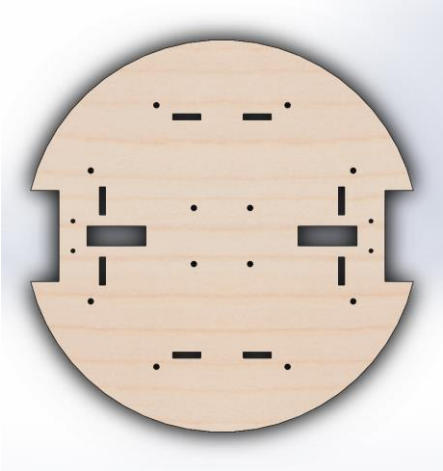

Figure 5: Top Side Laser Cutting Template.

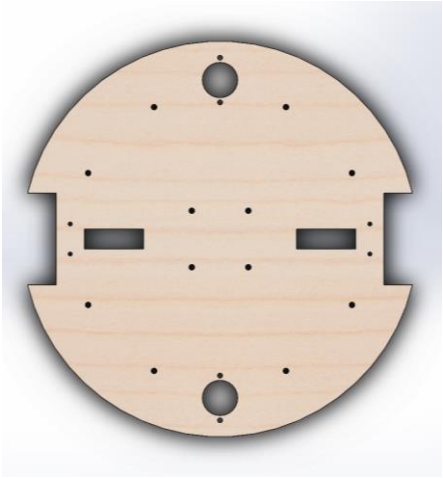

Figure 6: Bottom Side Laser Cutting Template.

The robot is driven by two 313 RPM gear motors with attached encoders (shown in Figure 7) centered on the left and right side of the base. The 2.975-inch polyurethane and nylon wheels provide a smooth and grippy connection to the floor. To support the robot on a total of four points, two ⅝ inch transfer ball casters are used in the front and rear of the robot. With this setup, the center of rotation of the robot is in the center of the circular base.

Figure 7: Wheel, Motor, and Encoder Assembly.

The robot is controlled using a Raspberry Pi 3 B (shown in Figure 8) and a RoboClaw 2×30A dual motor controller (shown in Figure 9). The Raspberry Pi was chosen due to the native wireless communication, the large processing power of 1.4 GHz, the USB inputs, and the GPIO (General Purpose Input/Output) pins [14]. Python 2.7 was used to program the Raspberry Pi, as it is a relatively simple language to learn and is also pre-built into the Raspberry Pi. The motor controller uses a USB connection to communicate with the Raspberry Pi, allowing for complex speed control to be handled on-board the RoboClaw, such as a PID controller. The RoboClaw has a Python library available to use. The manufacturer, Basic Micro, offers an external application to program and tune settings, such as PID controller in the motor controller [15].

Figure 8: Raspberry Pi Model B.

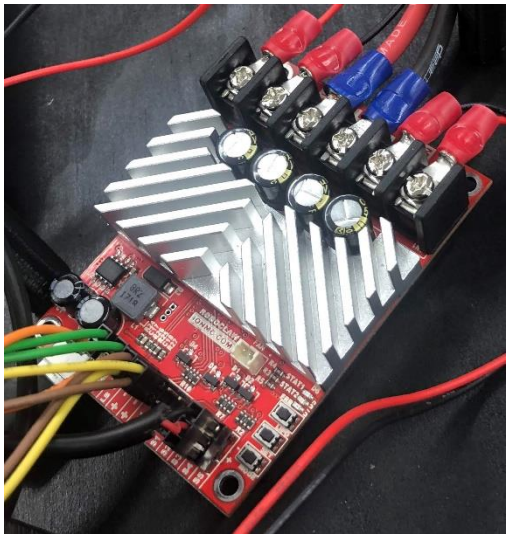

Figure 9: RoboClaw 2×30A Motor Controller.

Since the robotic base was changed from the previous MQP, the sensor platform also needed to be redesigned. The platform was designed to allow the center of the robot to have space available for the Raspberry Pi, IMU, and additional components. The sensor platform contains similar design characteristics to that of the stand from MAD 1802, such as the minimalist structure at the

top to reduce weight and the CO_2 sensor casing with exact sensor dimensions to allow for easy mounting. Another design characteristic is the ability to change the distances between the sensors. The stand is constructed using the same materials as the base, laser cut plywood and 3D printed parts, for ease of manufacturing, cost, and weight.

2.2 Analysis of Robot Design

To verify the capability of the new base design, several analyses were conducted. The top speed of the robot was calculated to be 0.78 m/s. From rest, the motors can provide up to 2.94 Nm of torque which is the total moment provided by the wheels (M_{wheel}). To prove the stability of the custom base, a simple summation of moments calculation was performed, as shown in Figure 10, Equation (2) and Equation (3). The calculations demonstrate that when the motors are at full torque, the robot still has 0.55 Nm of restoring torque (M_{net}) to keep it on the ground. The moments are calculated about A, the point at which one of the transfer caster balls touches the ground, where X_{CG} is the distance from A to the center of gravity, 0.1427m, and m is the mass of the robot, 2.5 kg.

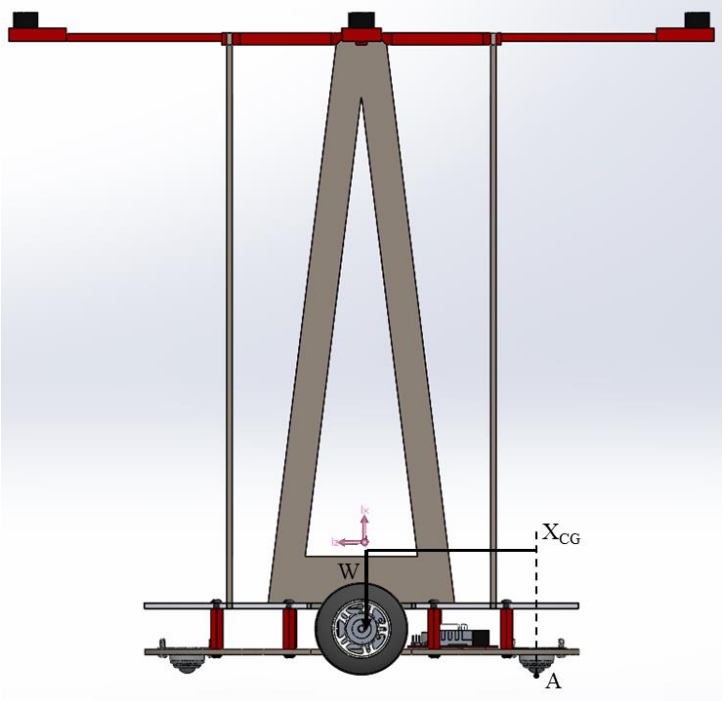

Figure 10: Schematic of Summation of Moments

$$\sum M_A = M_{net} = -M_{wheel} + X_{CG}mg \tag{2}$$

$$M_{net} = 0.55\ Nm \tag{3}$$

To ensure the sensor platform would not break under the load, simulations of the sensor platform in SolidWorks® were performed to determine the stresses and displacements. The maximum stress with the CO_2 sensors with a safety factor of two was well below the yield strength of the 3D printed plastic. The displacement on the end of the sensor arms was also very low, with a maximum of 0.4 mm as shown in Figure 11.

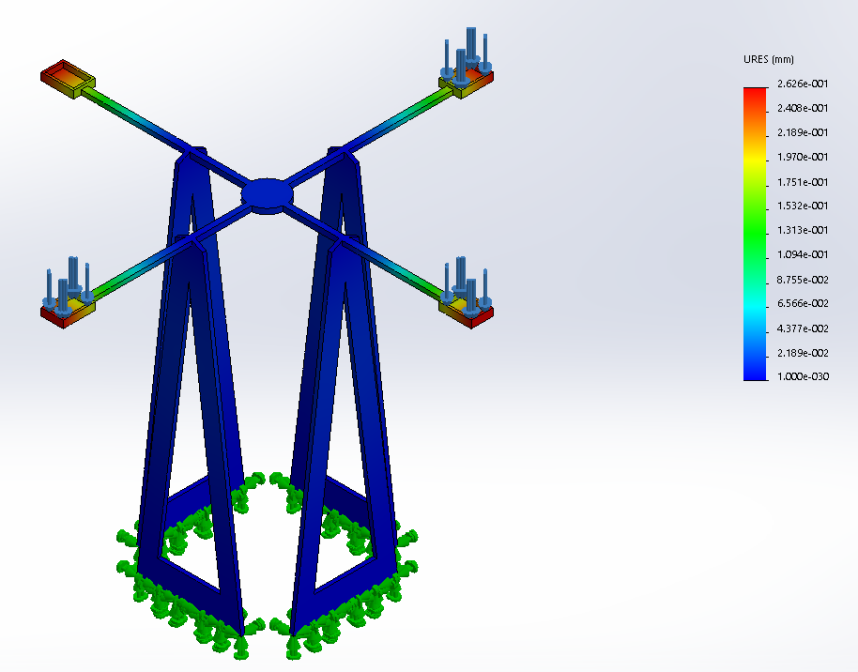

Figure 11: Displacement Analysis of the sensor platform in SolidWorks®.

2.3 Sensors used on robot

The robot uses several sensors to gather data about its movement and environment. The primary sensors are the four SprintIR WR 20% CO_2 Sensors (shown in Figure 12), which have a measurement range of 20% or 200000 ppm, a sampling rate of 20 Hz, (CO_2 meter), a rise time of about four seconds and a settling time of about 30 seconds (Figure 13). To connect the sensors to the Raspberry Pi, a USB to TTL cable and USB hub were used to easily communicate with sensors. In addition, the manufacturer's provided Python library was used to simplify the wiring.

Figure 12: SprintIR Sensor.

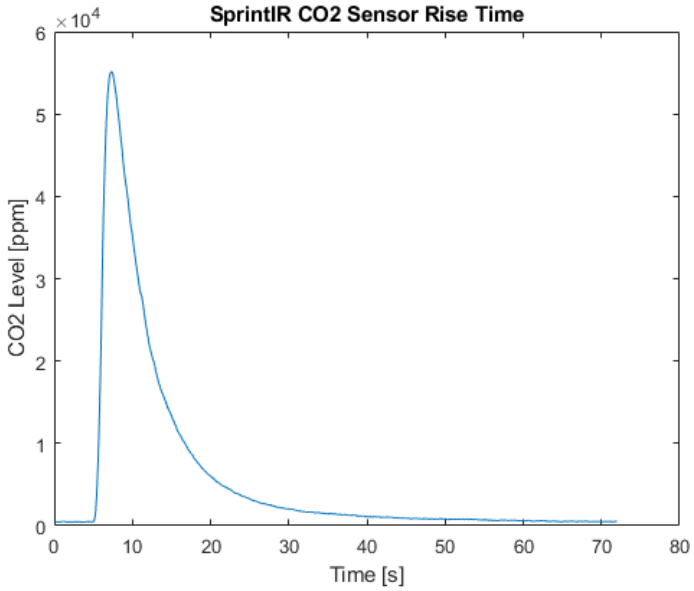

Figure 13: Rise and Settling time of SprintIR Sensor.

Another critical sensor on the robot is the inertial mass unit (IMU). The IMU is the Adafruit BNO055 Absolute Orientation Sensor (Figure 14), which is a nine degree of freedom IMU that measures three axes of acceleration, three axes of angular velocity, and heading with a magnetometer. This IMU also has a few built-in fusion algorithms, such as outputting Euler angles, quaternions, a gravity vector, and acceleration values with gravity removed. The supplier of the

IMU also provides a tutorial on using the IMU, as well as a Python library [16]. The IMU is connected to the Raspberry Pi using the I2C though the GPIO pins.

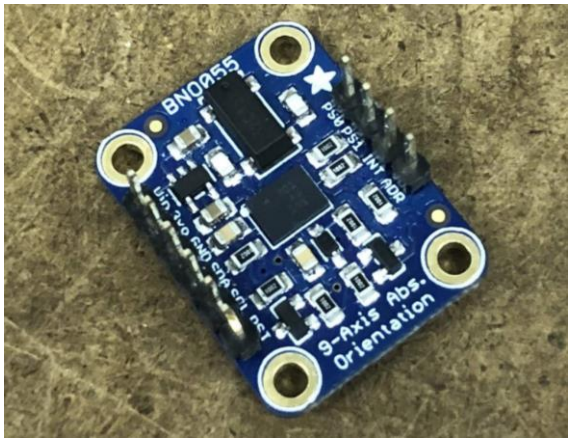

Figure 14: Adafruit BNO055 IMU.

The final sensors used for movement of the robot were the rotary quadrature magnetic encoders attached to the motors. The encoders are connected to the RoboClaw in order to perform PID control of the wheels. The encoder data can then be requested by the main controller through the USB connection of the RoboClaw.

2.4 Printed Circuit Board

A printed circuit board (PCB) was designed to help better integrate the IMU with the Raspberry Pi. The team used AutoDesk Eagle to design the PCB, seen in Figure 15. Once the design was finished, the board was laser cut and headers were soldered onto the PCB so it would fit both the IMU and the Raspberry Pi. The primary reason the PCB was designed was to decrease cluttering of the top of the robot and to produce a more visually appealing design. Moreover, it provided the team with an easier way to integrate the Raspberry Pi and IMU with the robot. However, due to complications in the manufacturing processes this PCB was not functional.

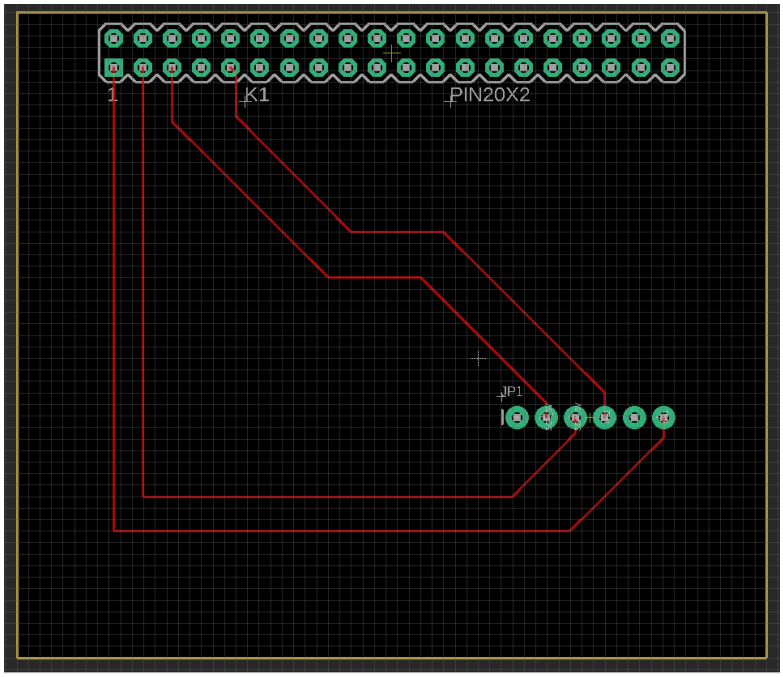

Figure 15: PCB Traces and Through Holes

3 Plume Generation System

The plume generation system, pictured in Figure 16, is used to produce the CO_2 for the robot to detect. The plume starts with a tank of CO_2 which then leads to a dual stage regulator. The tubing then connects the regulator to a solenoid valve. From there, the tubing leads to a flow meter and finally to the diffuser. The entire system is stationed on a stand that was built by a previous MQP team. The CO_2 tank is a 5-pound canister pressurized to approximately 700 psi. Since the pressure needs to be reduced for the gas to safely flow through the entire system, a dual stage regulator is used to decrease the pressure. After the pressure is decreased to about 100 psi, the CO_2 passes through the flow meter which is used to control the mass flow rate of the gas. The final piece of the system is the point source. The point source evenly diffuses the gas into the test area.

Figure 16: Plume Generation System.

3.1 Regulator

The Harris® Model 9296 Regulator, pictured in Figure 17, is a two-stage regulator that is used to decrease the pressure in a high-pressure tank to deliver a constant, specific pressure [6]. With a dual stage regulator, the CO_2 will enter the regulator and the pressure will be reduced to a fixed, intermediate value. Then, the second stage of the regulator will reduce the pressure to a desired and adjustable pressure. Between the first and the second stages the change in pressure is minor which allows the two stage regulator the ability to sustain a constant pressure without having to make adjustments [17].

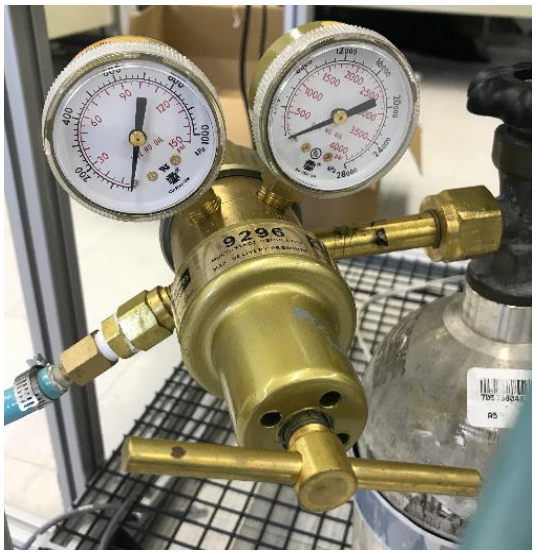

Figure 17: Harris® Model 9296 Regulator.

3.2 Flow Meter

The Sierra©SmartTrakC100L, shown in Figure 18, is used to set the mass flow of the CO_2. This flow meter is unaffected by upstream gas temperature and pressure, making it accurate and repeatable [18]. This device is able to determine the mass flow of the CO_2 by measuring the temperature change of the gas as it moves past a heated wire, which is connected to an internal PID- controlled valve [5]. The flow meter uses a 24V power source. Previous MQP teams have

determined that the best mass flow for this experiment would be 500 mg/s, but this flow meter will only work up to 459 mg/s [11].

Figure 18: Sierra© SmartTrak C100L.

3.3 Point Source

A point source is needed to evenly diffuse the CO_2 into the room. Previous MQP teams used a ping pong ball with poked holes as their point source, until last year's MQP designed a new point source that more evenly diffuses the CO_2. They designed the diffuser to have a chamber to allow the mean velocity of the gas to drop to about zero. From there the diffuser has two "arms" that lead the gas from the chamber to the point source, which is a sphere with holes in it (Figure 19).

Figure 19: Last Year's Diffuser.

The new point source is comprised of a sparger and 4-inch metal tubing, as shown in Figure 20. The main goal of the point source is to evenly diffuse CO_2 gas in a given area. Unlike last year's diffuser, the gas velocity is not impeded. This is because as long as the gas is able to uniformly distribute there is no need to reduce the speed. Additionally, the new diffuser is smaller and easier to manipulate when testing.

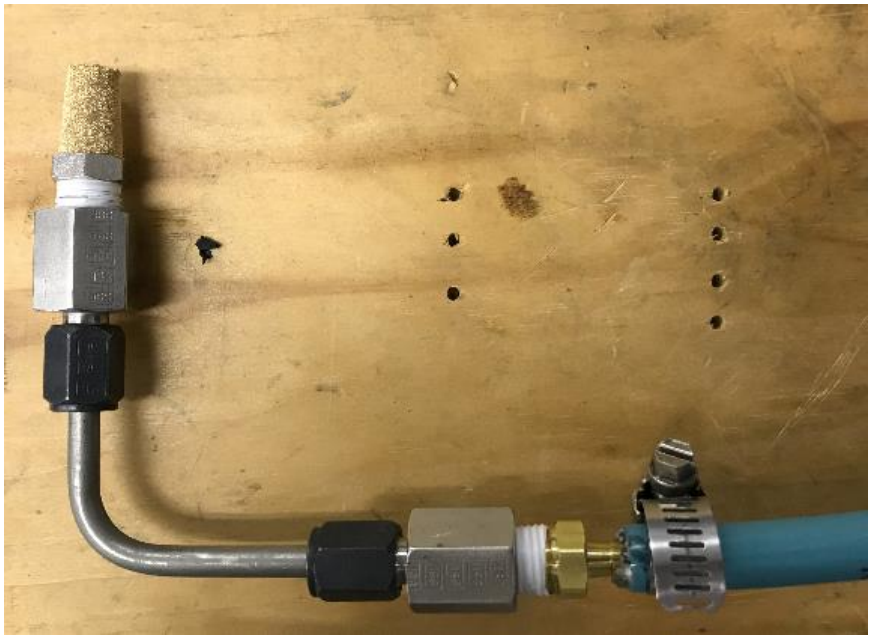

Figure 20: New Point Source Sparger.

4 Control

This section outlines the algorithm for controlling and tracking the robot. The CO_2 sensor readings and robot dimensions are used to move the robot by determining the plume heading and autonomously navigating in that direction. The EKF derivations are performed in this section to determine the equations that will allow the tracking and predicting of the robot state. This EKF prediction can be compared with the actual robot location found using an overhead camera and AprilTags to test its accuracy. More information about the control programs can be found in Appendix D.

4.1 Robot Movement

The robot uses data from the CO_2 sensors to determine where to navigate in order to arrive at the plume source. This control scheme assigns sensor 1 as the "lead sensor," meaning the robot will always turn so sensor 1 is heading towards the plume source. The CO_2 sensors begin emitting data in parts per million (PPM), and the Python script sorts these sensor values from greatest to least. In the schematic and in Equation (5), the sorted sensor values from greatest to least are labeled as "sorted[1]" through "sorted[4]." The program calculates the angle that the robot needs to turn based on sensor readings and the distance between sensors. Figure 21 depicts the four sensors (lead sensor is red), in a case where the right sensor is the highest and the top sensor is the second highest. The star denotes the location of maximum concentration.

D = distance between sensors (15.5 cm)
c = distance from robot center to sensor (11.25 cm)
a = distance from highest sensor to maximum PPM
b = distance from center to maximum PPM

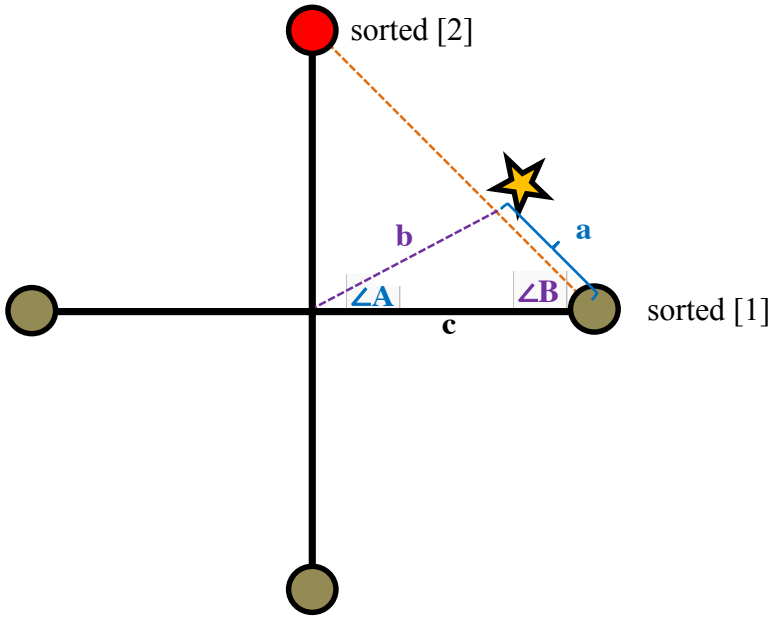

Figure 21: Robot Sensor Schematic.

The goal is to determine angle *A*, which can be found using the law of cosines in Equation (4),

$$A = \cos^{-1}\left(\frac{-a^2 + b^2 + c^2}{2bc}\right). \tag{4}$$

Distance *a* can be determined using the ratio of the sensor readings to total distance, as shown in Equation (5). The readings from the third highest sensor are subtracted from the first and second highest sensors in order to account for the baseline PPM reading. As a result, if the second and third highest sensor are approximately equal, the robot will move almost directly towards the sensor with the highest reading.

$$a = \frac{(sorted[2] - sorted[3])\ D}{\left((sorted[2] - sorted[3]) + (sorted[1] - sorted[3])\right)} \tag{5}$$

$$b = \sqrt{a^2 + c^2 - (2ac\cos(B))} \tag{6}$$

Angle *A* is always the angle from the maximum PPM reading to the highest sensor. The program then determines the total angle and direction the robot needs to turn so that the lead sensor is heading towards the maximum concentration. After the robot turns to that heading, it drives forward a set distance. The program then receives new PPM data and calculates a new angle to turn and drive towards. This process is repeated until the robot reaches the plume source. After the first rotation, the robot should only need to turn small angles left and right, since the plume source should be roughly straight ahead. The process is depicted in the pseudocode diagram shown in Figure 22.

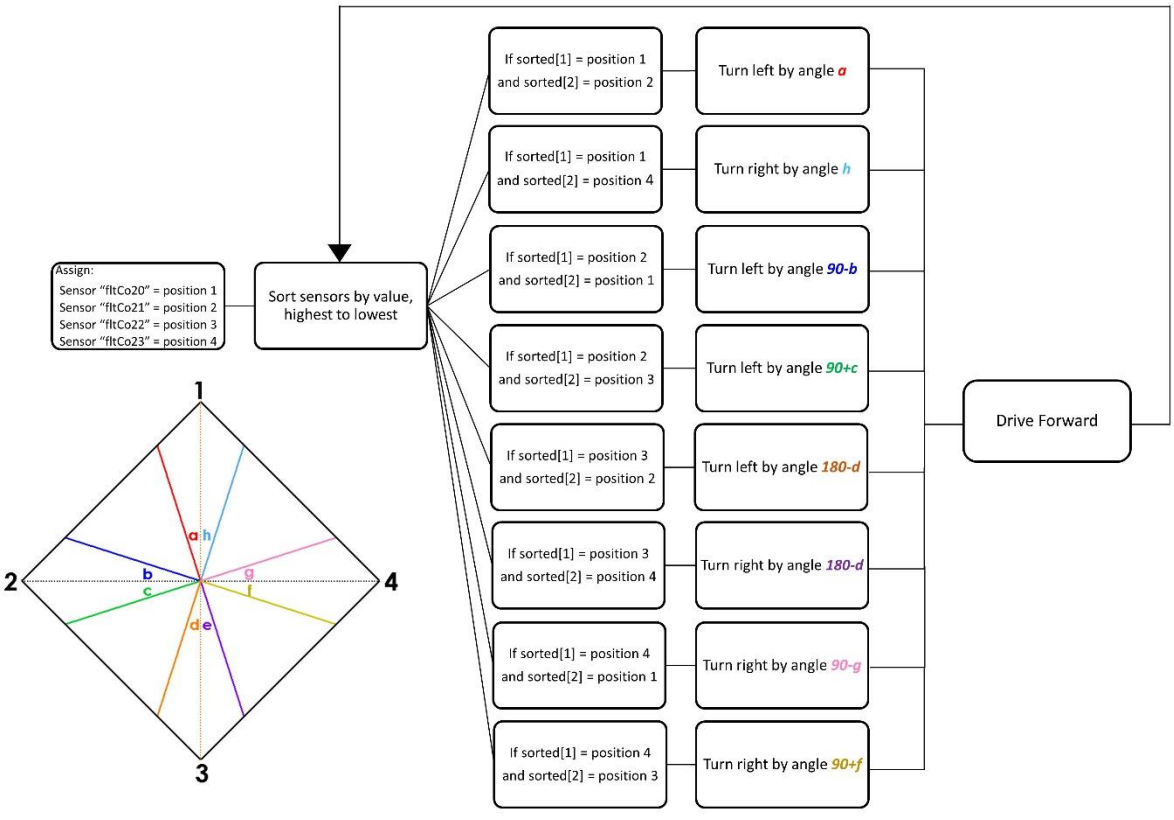

Figure 22: Robot Movement Pseudocode.

4.2 MATLAB Simulation of Searching Algorithm

To verify the accuracy of the robot movement algorithm and provide a visual representation of the robot path, the team created a MATLAB script for the experiment. It first simulates a CO_2 point source that has a strength inversely proportional to the distance squared, creating an X Y field of CO_2 values. To simulate noise within the field a small percentage perturbation is added to each grid point, the robot is then located at point i,j and the sensors located at i±1 and j±1. Next, the same robot movement algorithm from the Python script takes the CO_2 inputs from each sensor to rotate and move the robot accordingly. This results in a visual display of the robot navigating towards the plume source, accounting for slight variations and nonuniformity of the environment and plume source.

4.3 Kalman Filter

4.3.1 Kinematic Equations of Motion

The kinematic equations of motion are used to begin the derivation of the Kalman Filter equations needed to control the robot. These equations can only be used if constant acceleration is assumed and the velocities of the robot are relatively small. Because the robot is moving at slow speeds (~ 1 m/s) and has constant force from the motors creating a near-constant acceleration, the kinematic equations can be used. Figure 23 shows the inertial and body-fixed frames for the setup. The inertial frame is consistent with the position of the room so it does not change. The body-fixed frame is consistent with the position of the robot, and it will change as the robot moves.

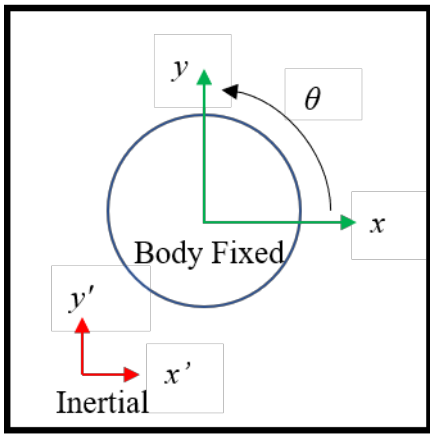

Figure 23: Body fixed and inertial fixed frames.

4.3.2 Time Discretization

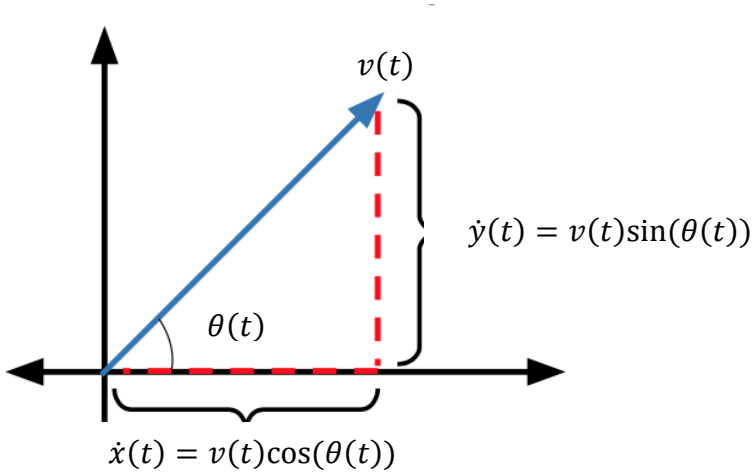

Figure 24: Components of Robot Velocity.

Equations (7), (8), and (9) are the kinematic equations of motion and the difference quotient for each time derivative $\dot{x}(t)$, $\dot{y}(t)$, $\dot{\theta}(t)$, and they show how the kinematic equation and the difference quotient can be approximately equal to each other. A visual representation of the time derivatives are shown in Figure 24. This takes the equations of motion from continuous to non-continuous time shown in Equations (7), (8), and (9)

$$\left.\begin{array}{c}\dot{x}(t) = v(t)\cos\big(\theta(t)\big) \\ \dot{x}(t) \approx \frac{x(t_n)-x(t_{n-1})}{\Delta T}\end{array}\right\} \Longrightarrow v(t)cos\big(\theta(t)\big) \approx \frac{x(t_n)-x(t_{n-1})}{\Delta T}, \tag{7}$$

$$\left.\begin{array}{c}\dot{y}(t) = v(t)\sin(\theta(t)) \\ \dot{y}(t) \approx \frac{y(t_n)-y(t_{n-1})}{\Delta T}\end{array}\right\} \Longrightarrow v(t)sin\big(\theta(t)\big) \approx \frac{y(t_n)-y(t_{n-1})}{\Delta T}, \tag{8}$$

$$\left.\begin{array}{c}\dot{\theta}(t) = \omega(t) \\ \dot{\theta}(t) \approx \frac{\theta(t_n)-\theta(t_{n-1})}{\Delta T}\end{array}\right\} \Longrightarrow \omega(t) \approx \frac{\theta(t_n)-\theta(t_{n-1})}{\Delta T}. \tag{9}$$

The time difference (ΔT) can be defined as the difference between the current time and the previous time shown in Equation (10)

$$\Delta T = t_n - t_{n-1}. \tag{10}$$

The current position is the sum of the previous position added to the velocity of the robot multiplied by the time step.

$$\text{current position} = \text{previous position} + (\text{previous velocity} \times \text{time})$$

The components of the current position are x_n, y_n, and θ_n. These are found using Equations (11), (12) , and (13) to find x_n, y_n, and θ_n, respectively

$$x(t_n) = x(t_{n-1}) + v(t_{n-1})\Delta T cos(\theta_n), \tag{11}$$

$$y(t_n) = y(t_{n-1}) + v(t_{n-1})\Delta T sin(\theta_n), \tag{12}$$

$$\theta(t_n) = \theta(t_{n-1}) + \omega(t_{n-1})\Delta T. \tag{13}$$

The velocity of the robot at the previous state is the average velocity of the left and right wheels at the previous state shown in Equation (14)

$$v(t_{n-1}) = \frac{v_L(t_{n-1}) + v_R(t_{n-1})}{2}. \tag{14}$$

The time is the time step, ΔT, which is equal to the time it takes the CO_2 sensors to update, as they are the slowest sensors at .05 seconds.

The angular velocity of the robot at the previous state is equal to the difference in velocities of the left and right wheels, divided by the distance between them shown in Equation (15)

$$\omega(t_{n-1}) = \frac{-v_L(t_{n-1}) + v_R(t_{n-1})}{W}. \tag{15}$$

The following notation, shown in Equations (16), (17), (18), (19) and (20) will be used for the rest of this paper

$$\begin{bmatrix} x(t_n) \\ y(t_n) \\ \theta(t_n) \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix}, \tag{16}$$

$$v(t_{n-1}) = v_{n-1}\,, \tag{17}$$

$$\omega(t_{n-1}) = \omega_{n-1}\,, \tag{18}$$

$$v_L(t_{n-1}) = v_{n-1}^L\,, \tag{19}$$

$$v_R(t_{n-1}) = v_{n-1}^R\,. \tag{20}$$

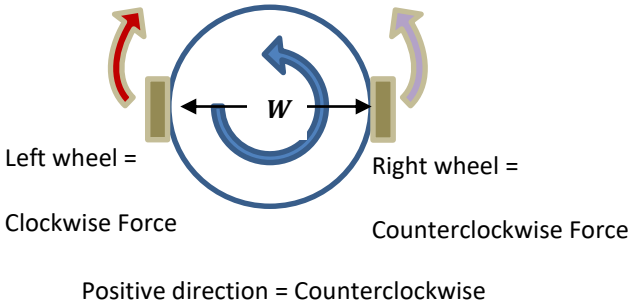

Figure 25: Acceleration and Angular Momentum.

A visual representation of the robot's acceleration and angular momentum is shown in Figure 25. Solving Equations (11), (12), and (13) using Equations (14) and (15), Equations (21), (22), and (23) can be determined as shown below

$$x_n = x_{n-1} + \frac{v_{n-1}^L + v_{n-1}^R}{2} \Delta T cos(\theta_n), \tag{21}$$

$$y_n = y_{n-1} + \frac{v_{n-1}^L + v_{n-1}^R}{2} \Delta T sin(\theta_n), \tag{22}$$

$$\theta_n = \theta_{n-1} + \frac{-v_{n-1}^L + v_{n-1}^R}{W} \Delta\,. \tag{23}$$

The current position, $\boldsymbol{x_n}$, can be described by a vector that consists of the position of $x(t)$, $y(t)$ and the angle $\theta(t)$

$$\boldsymbol{x_n} = \begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix}. \tag{24}$$

Using Equations (21), (22), and (23), the final discrete-time kinematic equations of the robot at time *n* can be represented in Equation (25)

$$\boldsymbol{x_n} = \begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix} = \begin{bmatrix} x_{n-1} + \frac{v_{n-1}^L + v_{n-1}^R}{2} \Delta T cos\left(\theta_{n-1} + \frac{-v_{n-1}^L + v_{n-1}^R}{W} \Delta T\right) \\ y_{n-1} + \frac{v_{n-1}^L + v_{n-1}^R}{2} \Delta T sin\left(\theta_{n-1} + \frac{-v_{n-1}^L + v_{n-1}^R}{W} \Delta T\right) \\ \theta_{n-1} + \frac{-v_{n-1}^L + v_{n-1}^R}{W} \Delta T \end{bmatrix}. \quad (25)$$

In order to accurately model these equations, the state-space representation is used. The three discrete time kinematic equations completely describe the motion of the robot. Therefore, x_{n-1}, y_{n-1}, and θ_{n-1} can be used as states in this model. The translational displacement $\boldsymbol{D}$ and angular displacement $\boldsymbol{\theta}$ are functions of the wheel velocities, v_L and v_R. The two equations below show the control inputs and the states respectively. Equation (26) defines u_n.

$$u_n = \begin{bmatrix} \boldsymbol{D} \\ \boldsymbol{\theta} \end{bmatrix} = \begin{bmatrix} \frac{v_{n-1}^L + v_{n-1}^R}{2} \Delta T \\ \frac{-v_{n-1}^L + v_{n-1}^R}{W} \Delta T \end{bmatrix}. \quad (26)$$

Using Equations (25) and (26), the next state, $\boldsymbol{x_n}$, can be found as a function of the current state, $\boldsymbol{x_{n-1}}$, and the inputs, u_n. The model accounts for process noise n_n added to the system. Equations (27) and (28) define the nonlinear dynamic model

$$\boldsymbol{x_n} = f(\boldsymbol{x_{n-1}}, u_n) + n_n\,, \quad (27)$$

$$f(\boldsymbol{x_{n-1}}, u_n) = \begin{bmatrix} x_{n-1} + \boldsymbol{D}\cos(\theta_{n-1} + \boldsymbol{\theta}) \\ y_{n-1} + \boldsymbol{D}\sin(\theta_{n-1} + \boldsymbol{\theta}) \\ \theta_{n-1} + \boldsymbol{\theta} \end{bmatrix}. \quad (28)$$

The Measurement Model Representation is based on the measurements taken from the 3-axis accelerometer and the 3-axis gyroscope, which are on the IMU. From these devices, only three measurements were needed: the x-axis acceleration, y-axis acceleration, and the angular rate in the

xy-plane, which is the rotation of the body with respect to the inertial frame. It is therefore necessary to convert these measurements into the same form as kinematic equations shown earlier. This is done using a "pseudo-integration" technique, whereby the x- and y-position are divided by ΔT^2, and the angular rate is divided by ΔT. The acceleration in the x- and y-direction and the angular rate are shown in Equations (29), (30), and (31)

$$a_{x,n} = \frac{x_n}{\Delta T^2}, \tag{29}$$

$$a_{y,n} = \frac{y_n}{\Delta T^2}, \tag{30}$$

$$\omega_n = \frac{\theta_n}{\Delta T}, \tag{31}$$

$$h(\boldsymbol{x_n}) = \begin{bmatrix} a_{x,n} \\ a_{y,n} \\ \omega_n \end{bmatrix} = \begin{bmatrix} \frac{1}{\Delta T^2} & 0 & 0 \\ 0 & \frac{1}{\Delta T^2} & 0 \\ 0 & 0 & \frac{1}{\Delta T} \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix}. \tag{32}$$

The nonlinear measurement model is shown in Equation (33). The nonlinear measurement model contains the function of the states at the next time step, denoted as $h(\boldsymbol{x_n})$, which is a nonlinear function. The measurement model also accounts for noise, denoted as w_n

$$z_n = h(\boldsymbol{x_n}) + w_n. \tag{33}$$

4.3.3 Linearization

The state Equation (25), must be linearized using the Jacobian, which is a matrix of all first-order partial derivatives of a function. The Jacobian of the state equation is given by equations (34) and (35)

$$\nabla_{x,y,\theta} f(\boldsymbol{x_{n-1}}) = \begin{bmatrix} \frac{\partial (f_1)_{n-1}}{\partial x_{n-1}} & \frac{\partial (f_1)_{n-1}}{\partial y_{n-1}} & \frac{\partial (f_1)_{n-1}}{\partial \theta_{n-1}} \\ \frac{\partial (f_2)_{n-1}}{\partial x_{n-1}} & \frac{\partial (f_2)_{n-1}}{\partial y_{n-1}} & \frac{\partial (f_2)_{n-1}}{\partial \theta_{n-1}} \\ \frac{\partial (f_3)_{n-1}}{\partial x_{n-1}} & \frac{\partial (f_3)_{n-1}}{\partial y_{n-1}} & \frac{\partial (f_3)_{n-1}}{\partial \theta_{n-1}} \end{bmatrix}, \tag{34}$$

$$\nabla_{x,y,\theta} f(\boldsymbol{x_{n-1}}) = \begin{bmatrix} 1 & 0 & -\boldsymbol{D} sin(\theta_{n-1} + \boldsymbol{\theta}) \\ 0 & 1 & \boldsymbol{D} cos(\theta_{n-1} + \boldsymbol{\theta}) \\ 0 & 0 & 1 \end{bmatrix}. \tag{35}$$

Similarly, the Jacobian of the linearized state matrix is shown in Equations (36) and (37), using Equation (28).

$$\nabla_{\boldsymbol{D},\boldsymbol{\theta}} f(\boldsymbol{x_{n-1}}, u_n) = \begin{bmatrix} \frac{\partial (f_1)_{n-1}}{\partial \boldsymbol{D}} & \frac{\partial (f_1)_{n-1}}{\partial \boldsymbol{\theta}} \\ \frac{\partial (f_2)_{n-1}}{\partial \boldsymbol{D}} & \frac{\partial (f_2)_{n-1}}{\partial \boldsymbol{\theta}} \\ \frac{\partial (f_3)_{n-1}}{\partial \boldsymbol{D}} & \frac{\partial (f_3)_{n-1}}{\partial \boldsymbol{\theta}} \end{bmatrix} \tag{36}$$

$$\nabla_{\boldsymbol{D},\boldsymbol{\theta}} f(\boldsymbol{x_{n-1}}, u_n) = \begin{bmatrix} -\boldsymbol{D} sin(\theta_{n-1} + \boldsymbol{\theta}) & cos(\theta_{n-1} + \boldsymbol{\theta}) \\ \boldsymbol{D} cos(\theta_{n-1} + \boldsymbol{\theta}) & sin(\theta_{n-1} + \boldsymbol{\theta}) \\ 1 & 0 \end{bmatrix} \tag{37}$$

Similar to the previous equations, Equations (38) and (39) are the Jacobian of the measurement function. As seen in Equation (39), it does not differ from Equation (32) since $h(\boldsymbol{x_n})$ is not a function of time

$$\nabla_{x,y,\theta} h(\boldsymbol{x_n}) = \begin{bmatrix} \frac{\partial (h_1)_n}{\partial x} & \frac{\partial (h_1)_n}{\partial y} & \frac{\partial (h_1)_n}{\partial \theta} \\ \frac{\partial (h_2)_n}{\partial x} & \frac{\partial (h_2)_n}{\partial y} & \frac{\partial (h_2)_n}{\partial \theta} \\ \frac{\partial (h_3)_n}{\partial x} & \frac{\partial (h_3)_n}{\partial y} & \frac{\partial (h_3)_n}{\partial \theta} \end{bmatrix}, \tag{38}$$

$$\nabla_{x,y,\theta} h(\boldsymbol{x_n}) = \begin{bmatrix} \frac{1}{\Delta T^2} & 0 & 0 \\ 0 & \frac{1}{\Delta T^2} & 0 \\ 0 & 0 & \frac{1}{\Delta T} \end{bmatrix}. \tag{39}$$

Equations (40) through (42) show the A_n, B_n, and C matrices as functions of the Jacobians

$$\nabla_{x,y,\theta} f(\boldsymbol{x_{n-1}}) = \frac{df_{n-1}}{d\boldsymbol{x_{n-1}}}\Big|_{\boldsymbol{x_{n-1}}} \equiv A_n \, , \tag{40}$$

$$\nabla_{\boldsymbol{\theta},\boldsymbol{D}} f(\boldsymbol{x_{n-1}}, u_n) = \frac{df_{n-1}}{du}\Big|_{\boldsymbol{x_{n-1}}, u_n} \equiv B_n \, , \tag{41}$$

$$\nabla_{x,y,\theta} h(\boldsymbol{x_n}) = \frac{dh_n}{dx}\Big|_{\boldsymbol{x_n}} \equiv C. \tag{42}$$

4.3.4 Prediction and Update

The motion of this robot cannot be described as a linear system. Therefore, an EKF must be used to estimate the system states. The EKF expands upon the Kalman Filter with an added step to linearize the predicted state estimates. Due to approximations in linearization, the EKF is not an optimal estimation scheme, but rather an approximation of one. Upon determining the linear dynamic model and linear measurement model, Equations (43) through (49) are used to determine the estimated state of the robot.

Prediction Phase:

Predicted state estimate

$$\widehat{\boldsymbol{x}}_{\boldsymbol{n}}^{-} = f(\boldsymbol{x}_{\boldsymbol{n-1}}, u_n) \tag{43}$$

$$z_n = C\boldsymbol{x}_{\boldsymbol{n-1}} + w_{n-1} \tag{44}$$

Predicted covariance estimate

$$P_n^- = A_n P_{n-1} A_n^T + Q_n \tag{45}$$

Update Phase:

Kalman gain

$$K_n = P_n^- C^T R^{-1} \tag{46}$$

Measurement noise covariance

$$R = \begin{bmatrix} \sigma_{ax}^2 & 0 & 0 \\ 0 & \sigma_{ay}^2 & 0 \\ 0 & 0 & \sigma_g^2 \end{bmatrix} \tag{47}$$

Updated covariance estimate

$$P_n = (I - K_n C)\, P_n^- \tag{48}$$

Updated state estimate

$$\hat{x}_n = \hat{x}_n^- + K_n(z_n - C\hat{x}_n^-) \tag{49}$$

4.4 Extended Kalman Filter Implementation

With the Extended Kalman Filter fully defined for the equations of motion of the robot, it could then be implemented into the robot's navigation Python script. The EKF function takes in new data every iteration of the searching algorithm, defined in section 4.1. Feeding the function reasonable and consistent values the EKF did output logical location data. However due to time and software errors, the function never produced logical data in real time.

4.5 AprilTags

AprilTags are two-dimensional barcodes that can be located with the use of a camera and the AprilTags detection software [19]. This software was run on an Ubuntu Virtual Machine. To track the robot throughout the experiment, AprilTags, the team planned to use a vision processing program. AprilTags is based on OpenCV and uses QR code-like images to determine the position and pose of the object in space. The AprilTags program would send position data to the robot to update and verify the location of the EKF. AprilTags requires a Linux system to run, which was achieved by setting up a virtual machine and attaching a USB webcam to the ceiling of the experimental area. Figure 26is a result of running the example program by following steps listed published by MIT [20].

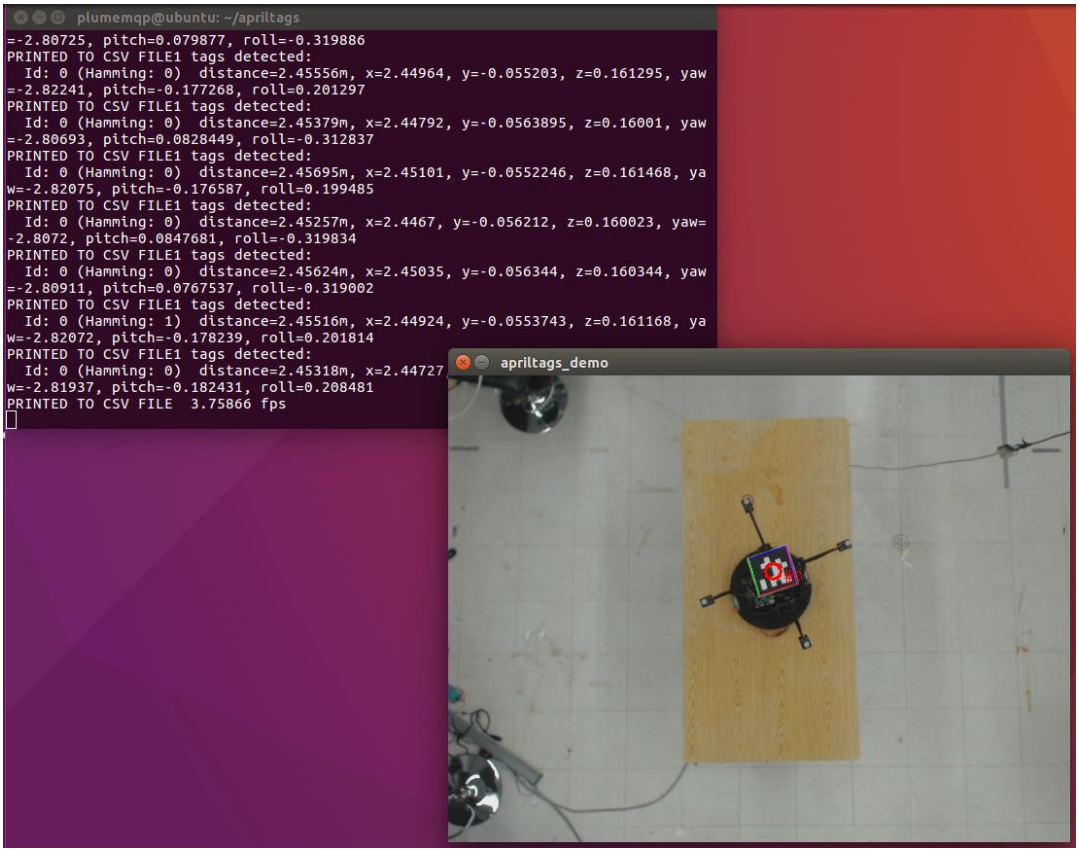

Figure 26: AprilTags Demo Locating an AprilTag.

For this project, Tag 36h11 was printed and attached to the sensor platform of the robot. Additionally, to ensure that the experiment did not take place out of view of the camera, the area of the experimentation field, Equation (50), was calculated using W_{Lens}, the width of the lens and L_{Focal}, the focal length.

$$A = 2\tan^{-1}\left(\frac{W_{lens}}{2\,L_{focal}}\right) \tag{50}$$

Due to software difficulties and time limitation, AprilTags were not fully implemented into the experiment.

5 Summary, Conclusions, Recommendations

5.1 Summary

This project culminated many different aspects of engineering, from robotic design and control to creating a functional plume generation system that behaves as an accurate point source. For the design of the robot, the team utilized SolidWorks® to design and create a model for parts that could be manufactured in a 3D printer and laser cutter. Motors, a motor controller, a Raspberry Pi, CO_2 sensors, and an IMU were then added to the robot skeleton to create the final design. The motors and motor controller were used for the movement of the robot, and the CO_2 sensors and IMU were used to control the robot.

Utilizing the control sensors (IMU and CO_2 sensors), a robot guidance system was created that ran on the Raspberry Pi. The guidance system was written in Python and used the measurements from the sensors to navigate the robot towards the highest CO_2 concentration. Also, the team built an EKF algorithm to minimize errors in the guidance system. The EKF received data from the sensors, and paired that with the previous states position, velocity, and acceleration to accurately predict the current state of the robot. Another way to minimize the errors in the guidance system is using an AprilTags camera to track the robot's position.

The final portion of the project was designing a plume generation system that accurately models a point source of CO_2. The components to the plume system include a regulator, flow meter, and sparger. The team used the previous MQP's plume system model to begin the design, after running into multiple problems, the system was made much simpler and easier to operate.

After the design process was finished, experimentation of the guidance system needed to be performed to ensure the system was working correctly. Once the guidance system was functioning

correctly, the team worked on experimenting with the AprilTags and EKF to eliminate errors from the guidance system.

5.2 Conclusions

The project challenge was to develop a plume generation system, develop a robot capable of autonomously locating the CO_2 plume source, and track the robot's position using an Extended Kalman Filter. The team was able to redesign the plume generation system. This was done by removing the solenoid valve to allow the CO_2 to flow through the system. Also, the point source was replaced with a sparger to more evenly and effectively diffuse the CO_2. When designing the new diffuser, the team had the opportunity to learn about the flow of the system and how to effectively model a point source. Additionally, when addressing the solenoid valve issue, the team worked with and built circuit boards in an attempt to open the valve. Many tests were run to discover what was wrong with the valve. In the end, the team determined that the circuit they made was faulty and due to a lack of time and readily available resources the team decided to remove the solenoid valve.

The team successfully designed and programmed a robot to navigate towards a CO_2 plume source. This was accomplished by developing and implementing an algorithm that enabled the robot to turn and drive forward towards the highest CO_2 concentration. Lessons learned included how to write a Python script that commands the robot to move and how to develop an algorithm that uses CO_2 data as inputs, in a Python script. After observing the robot move towards the plume source and receiving feedback from the advisor, it became apparent that there are many ways to solve this problem, some which may be more efficient than the one outlined in this paper. In addition, another important lesson learned was that the sensors have a long settling

time, which greatly affects the robot navigation. This taught the team that the sensors chosen have a substantial impact on the capabilities of the robot.

Derivation of the Extended Kalman Filter equations taught the team how vehicles are capable of autonomous navigation based on inputs from sensor measurements. Once again, accuracy of sensor measurements proved to have an important impact on the accuracy of the Extended Kalman Filter. Sanity checks to make sure the position and acceleration measurements are reasonable were performed to check the plausibility of the measurements. The team soon learned that this algorithm is not a perfect solution; small errors and variations in the observation or measurements can cause the EKF to inaccurately estimate the robot's position.

5.3 Recommendations for Future Work

This section provides recommendations for advancing the methods, experiments, and work performed throughout this project. These recommendations include accounting for the high settling time of the CO_2 sensors, continuing the work with EKF and AprilTags tracking, developing a more efficient movement scheme to find the plume source, and creating a more advanced system for the robot connection and communication system.

5.3.1 Account for High Settling Time of CO_2 Sensors

One problem the team encountered when executing the Python script was that the settling time for the sensors was very large (~30 seconds). As a result, after the robot would turn and drive forward, the same two sensors were often still high so the robot would turn to their new location, away from the plume source. This can be resolved by adding a long delay (~30-60 seconds) between movements; however, there are more appropriate and fitting solutions that will not increase the time to find the plume source by as much. One potential solution is to determine the increase in CO_2 during the previous couple of seconds and drive towards the sensors with the most

increase. Therefore, even if the other sensors are still high, if they did not recently experience an increase the robot will not mistakenly navigate in their direction.

5.3.2 EKF and AprilTags Testing and Comparison

Another recommendation is to continue work on the EKF and AprilTags tracking of the robot. The virtual machine for AprilTags is enabled so that the camera can view the robot path. The next step is to import the data from the path of the robot into MATLAB. The EKF provides an accurate estimation of the (x, y, θ) coordinates, but the other equations should be verified. After the numbers are determined to be reasonable, the estimated route by the EKF and true path observed by the camera can be plotted on the same graph to determine the error of the EKF.

5.3.3 More Efficient Path to Plume Source

A third recommendation that can improve the project is the method of moving towards the plume source. To simplify the method of movement, this robot has isolated turn and drive forward movements. A more efficient way for the robot to move may be to drive and turn at the same time, so time is not wasted at the same (x, y) coordinates. This will require more complex mathematical equations, as well as a more difficult coding scheme for execution. To further decrease the time to find the plume source, a robot with omnidirectional wheels may be utilized so that the robot can travel towards the highest concentration without needing to physically turn the whole robot.

5.3.4 A Robust and Consistent means to Connect the Robot

A final recommendation is creating a more robust and consistent way of connecting and running the robot. Through the course of this project, the Raspberry Pi had a dynamic IP. This meant that any time the Raspberry Pi was powered up, the IP address would be different than a previous time the robot was run. This can be resolved by requesting a static IP from WPI

NetOpps or operating the robot on a separate network where the IP address can be assigned. To run the programs on the robot, a simple SSH (secure shell) connection was used. SSH is a simple way to access the terminal of the Raspberry Pi. However, for a more polished performance, it would be useful to implement a web socket server to run commands from as well as a cleaner looking way to output data.

6 References

[1] “Climate change causes: A blanket around the Earth,” *NASA*Available: https://climate.nasa.gov/causes/.

[2] Gohd, C., “Self-driving rovers could be the future of exploration on Mars,” *Astronomy.com* Available: http://www.astronomy.com/news/2019/01/self-driving-rovers-could-be-the-future-of-exploration-on-mars.

[3] Grewal, M. S. (2010, May 18). Applications of Kalman Filtering in Aerospace 1960 to the Present [Historical Perspectives]. Retrieved from https://ieeexplore.ieee.org/document/5466132

[4] McGee, L. A. (1985, November). Discovery of the Kalman Filter as a Practical Tool Retrieved from https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19860003843.pdf

[5] Bakker, A. (2006). Kolmogorov's Theory. Retrieved from http://www.bakker.org/dartmouth06/engs150/09-kolm.pdf

[6] Clark, C., Greene, M., & Seigel, M. (2015, March 23). *Design of Plume Generation and Detection System*[Scholarly project]. Retrieved October 11, 2018.

[7] Anglin, M., Hunt, M., & Myles, M. (2015, March 27). *Gas Source Localization with a Mobile Sensing Ground Vehicle*[Scholarly project]. Retrieved October 11, 2018.

[8] Barney, M., & Rivary, S. (2016, March 4). *Design and Integration of an Indoor Plume Experimental Setup* [Scholarly project]. Retrieved October 10, 2018.

[9] Christie, N., & Colfer, J. (2016, March 25). *Plume Estimation Using a Gas-Sensing Mobile Robot*[Scholarly project]. Retrieved October 11, 2018.

[10] Fast, E., Harnais, S., & Weisenberg, R. (2017, March 3). *Plume Analysis and Detection*[Scholarly project]. Retrieved October 11, 2018.

[11] Halama, S., Kasapis, S., Kontopyrgos, M., McGrath, O., & Preston, B. (2018, March 2). *Estimation of a Plume with an Unmanned Terrestrial Vehicle*[Scholarly project]. Retrieved October 11, 2018.

[12] Grémillet, D., Puech, W., Garçon, V., Boulinier, T., and Maho, Y. L., “Robots in Ecology: Welcome to the machine,” *Open Journal of Ecology*, vol. 02, 2012, pp. 49–57.

[13] “Climate change causes: A blanket around the Earth,” *NASA*Available: https://climate.nasa.gov/causes/.

[14] “Raspberry Pi 3 Model B ,” *Rotate display 90º? - Raspberry Pi Forums*Available:https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/.

[15] “Roboclaw 2x30A Motor Controller ,” *ServoCity.com* Available: https://www.servocity.com/roboclaw-2x30a-motor-controller.

[16] “Adafruit BNO055 Absolute Orientation Sensor,” *Memory Architectures | Memories of an Arduino | Adafruit Learning System*Available: https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/overview.

[17] Dual Stage Regulators. (n.d.). Retrieved October 10, 2018, from https://store.mathesongas.com/dual-stage-regulators/

[18] Premium Digital Mass Flow Controllers and Mass Flow Meters. (n.d.). Retrieved October 10, 2018, from http://www.sierrainstruments.com/products/c100.html

[19] “AprilTags C Library,” *Convection From a Rectangular Plate*Available: http://people.csail.mit.edu/kaess/apriltags/.

[20] “AprilTag,” *AprilTag*Available: https://april.eecs.umich.edu/software/apriltag.html.

7 Appendices

Appendix A

Component	Specification	Source
12 volt 313 rpm gear motor with encoder	Speed (No Load) - 313 rpm Current (No Load) - 0.52A Current (Stall) - 20A Torque (Stall) - 416.6 oz-in Gear Ratio - 26.851:1 Encoder – Magnetic Relative Quadrature	ServoCity
2.975 inch wheel	Coefficient of friction ≅ 1	ServoCity
Roboclaw 2×30A Motor Controller	5V 3A Switch Mode BEC USB interface Quadrature Encoder input Position and Velocity Control	ServoCity
SprintIR Wide Range 20% CO2 Sensor	Measurement range- 20% (20,000ppm) Accuracy- ±70 ppm ± 5% of reading Power Input: 3.2-5.5VDC Peak current: 100mA Average Current: <15mA	CO2 Meter
Raspberry Pi 3 B+	1.4GHz 64-bit quad-core processor, dual-band wireless LAN, Bluetooth 4.2/BLE.	

Appendix B

This appendix outlines how to calculate the sensor bias for the IMU. First, a series of tests must be conducted with the to determine measured values at each state. Then, these are compared to the known values to calculate the sensor bias.

1. Collect data for 60 seconds for each of the following cases:
    a. Lay the IMU on a flat surface. Check with a liquid level.
    b. Mount the IMU vertically on one edge. Check with a liquid level.
    c. Mount the IMU vertically on its other edge. Check with a liquid level.
    d. Lay the IMU flat on a turntable. Check with a liquid level.
    e. Mount the IMU vertically on one edge on a turntable. Check with a liquid level.
    f. Mount the IMU vertically on its other edge on a turntable. Check with a liquid level.
2. Average the data for each of the cases to find the measured averages (μ).
3. Determine the true values for each of the cases.
4. Calculate the sensor bias using the equation:

$$b = true\ value - \mu$$

Appendix C

This appendix outlines how to setup the Raspberry Pi to the WPI wireless Network.

1. First, make sure the MAC address of the Raspberry Pi is known.
2. Next, take the Raspberry Pi to the IT help desk in the library and obtain a ticket from them to go to WPI Network Operations (NetOps).
3. Once you arrive at NetOps, ask for assistance connecting the Raspberry Pi to the wireless network.

Appendix D

The Python and MATLAB scripts can be found on the following GitHub link https://github.com/dcweber765/PlumeMQP.

The Python scripts range from testing individual components to the integrated robot system.